CSC447 Machine Learning Final Project

Wine Quality Classification Prediction

Alice Liu
23831497
aliu005@citymail.cuny.edu

The City College of New York
Grove School of Engineering, Computer Science

Introduction

Winemaking is revered as an art form by consumers and connoisseurs alike, where each bottle produced, embodies a unique and complex array of flavors, aromas, and experience. This sensory journey can be considered not purely subjective. The intricate chemistry of wine governs its taste, texture, and overall quality, making it a captivating subject for scientific exploration.

By leveraging machine learning techniques, we aim to construct a predictive model capable of discerning wine quality based on a myriad of chemical attributes. In this report, I will outline the methodologies employed to the development of this project. This includes the collection of data, preprocessing, exploratory data analysis (EDA), modeling, feature selection, and model evaluation.

Dataset

The dataset utilized for this project can be accessed on Kaggle via the following link: Wine Quality Dataset

To gain a visual understanding of the dataset's structure, we will examine the first 5 rows using `df.head()`. The representation of the dataset is illustrated in Figure 1A below:

```
In [2]: df = pd.read_csv("data/WineQT.csv", header = 0)
        df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	2
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	3
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	4

Figure 1A. Dataset Structure - Using df.head()

Initial exploration of this dataset involves uncovering its contents and structure. Key aspects we aim to understand include:

- **Data Shape**: discover how much data we have. Do this by determining the dataset's size in terms of rows and columns. By executing `df.shape` in the code block, we obtain a tuple whose syntax is (x, y), where `x` represents the number of columns and `y` represents the number of rows. This information provides a fundamental understanding of the dataset's dimensions and scope.

- **Column Names**: identifying the column names is crucial. Executing `df.columns` in the code block provides a list of column names, offering insight into the available data fields.

- **Column Datatype**: for a deeper understanding of the dataset's columns, executing`df.dtypes` in the code block provides the datatype of each column in the dataframe. This enables a comprehensive overview of the data's structure and facilitates informed data handling and analysis.

- **NULL Information**: identifying and managing missing values is crucial, particularly in machine learning and the development of prediction models. By executing `df.isnull().sum()` in the code block, we obtain a summary of the number of NULL values present in each column of the dataframe.

- **Summary Statistics**: executing `df.describe()` generates descriptive statistics for numerical columns in the dataframe, including count, mean, standard deviation, minimum, quartiles (25%, 50%, 75%), and maximum values. This summary provides valuable insights into the distribution, central tendency, and variability of numerical data, aiding in the initial assessment and understanding of key features within the dataset.

- **Unique Values**: to identify unique values within specific columns, execute `df[col_name].unique()`. This command returns an array containing distinct values present in the specified column, providing insight into the unique data points and their distribution within the dataset.

These preliminary steps lay the foundation for a deeper exploration and analysis of the dataset. This exploration is highlighted in Figure 1B, Figure 1C, and Figure 1D.

```
In [4]: print("Shape:", df.shape)
        print("--------------------------------------------------")
        print("Columns:")
        print(df.columns)
        print("--------------------------------------------------")
        print("Datatype:")
        print(df.dtypes)
        print("--------------------------------------------------")
        print("NULL information:\n")
        print(df.isnull().sum())
        print("--------------------------------------------------")
```

```
Shape: (1143, 13)
--------------------------------------------------
Columns:
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
--------------------------------------------------
Datatype:
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
Id                        int64
dtype: object
--------------------------------------------------
NULL information:

fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
--------------------------------------------------
```

Figure 1B. Understanding the Dataset Structure - shape, column, datatype, and NULL

```
In [4]: df.describe()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015	0.657708	10.442111	5
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.170399	1.082196	0
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.550000	9.500000	5
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.620000	10.200000	6
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.730000	11.100000	6
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8

Figure 1C. Understanding the Dataset Structure - statistical summary

```
In [5]: df["quality"].unique()
Out[5]: array([5, 6, 7, 4, 8, 3])
```

Figure 1D. Understanding the Dataset Structure - unique value of column `quality`

```
In [6]: # analyze distribution of unique values from column quality
        plt.figure(figsize=(8, 6))
        value_counts = df["quality"].value_counts()
        sns.barplot(x=value_counts.index, y=value_counts.values, palette="viridis")

        plt.xlabel("Quality", fontsize=12)
        plt.ylabel("Count", fontsize=12)
        plt.title("Distribution of Wine Quality", fontsize=14)
        plt.show()
```

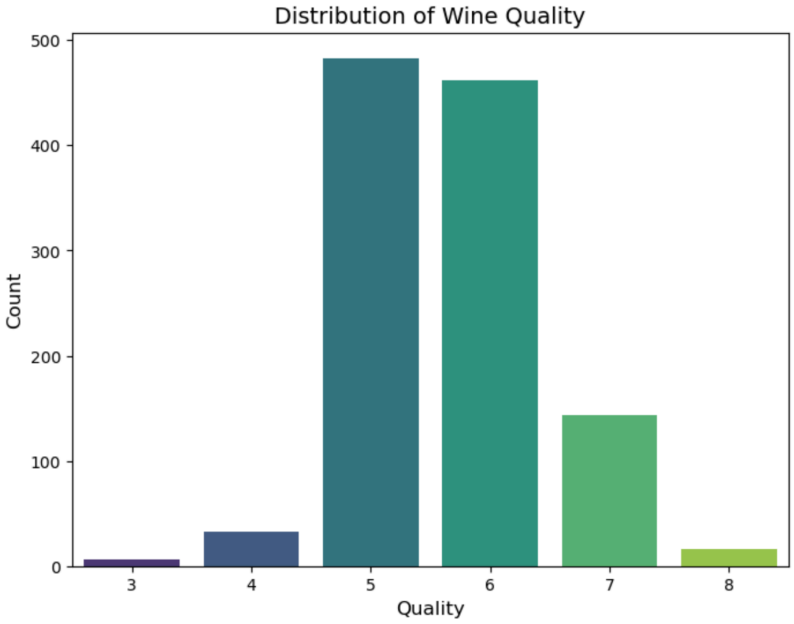


Figure 1E. Understanding the Dataset Structure - unique value of column `quality`

From Figures 1B, 1C, and 1D we obtain the following insights:

- **Data Shape**: (1143, 13) meaning 1143 columns and 13 rows

- **Column Names**: 13 columns: ‘fixed acidity’, ‘volatile acidity’, ‘citric acid’, ‘residual sugar’, ‘chlorides’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘density’, ‘pH’, ‘sulphates’, ‘alcohol’, ‘quality’, ‘Id’.

  Target variable: quality

- **Column Datatype**: all columns have datatype: float64. Only column `quality` and `Id` have datatype: int64.

- **NULL Information**: no NULL values

- **Summary Statistics**: the count for each column is consistent at 1143 (same as column value of df), indicating that there are no missing values in these numerical features.

- **Unique Values**: I examined the unique values within the `quality` column to gain insight into the criteria defining whether a specific wine was deemed good or bad. The unique values observed

are 3, 4, 5, 6, 7, and 8. 8 indicates a higher rating and 3 indicates a lower rating of quality. From Figure 1E, we can observe the distribution of these unique values. Notably, there is a predominant distribution of wines rated with a quality of 5 and 6, suggesting that these quality ratings are most prevalent within the dataset.

```
In [6]: # DROP column Id
        df.drop(['Id'], axis = 1, inplace = True)
```

Figure 1F. Drop Column `Id`

Upon analysis, I determined that the `Id` column was redundant as its values sequentially mirrored the row indices from 0 to 1143, essentially serving as an identifier without providing additional meaningful information. Consequently, I removed the `Id` column from the dataset using the code block demonstrated in Figure 1F. Subsequently, the revised dataset `df` is depicted in Figure 1G.

```
In [7]: df.head()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Figure 1G. New Dataset Structure - After Dropping Column `Id`

Exploratory Data Analysis (EDA)

I conducted an analysis to identify and assess the presence of outliers within each column of the dataset. This exploration is crucial because outliers have the potential to significantly impact the performance, accuracy, and reliability of predictive models. When a column contains extreme outliers, the mean may not accurately reflect the central tendency of the data, which can result in misleading interpretations and predictions. Moreover, outliers can significantly impact error rates, particularly in classification tasks.

To visualize and pinpoint outliers effectively, I utilized box plots, as depicted in Figure 2A, which illustrates the distribution and variability of each column's data. In a box plot, outliers are represented as individual data points beyond the whiskers of the plot, often depicted as circles. This visualization method provides a clear and intuitive way to identify any potential outliers across the dataset, enabling informed decisions on outlier treatment strategies to enhance the robustness and reliability of analysis and modeling.

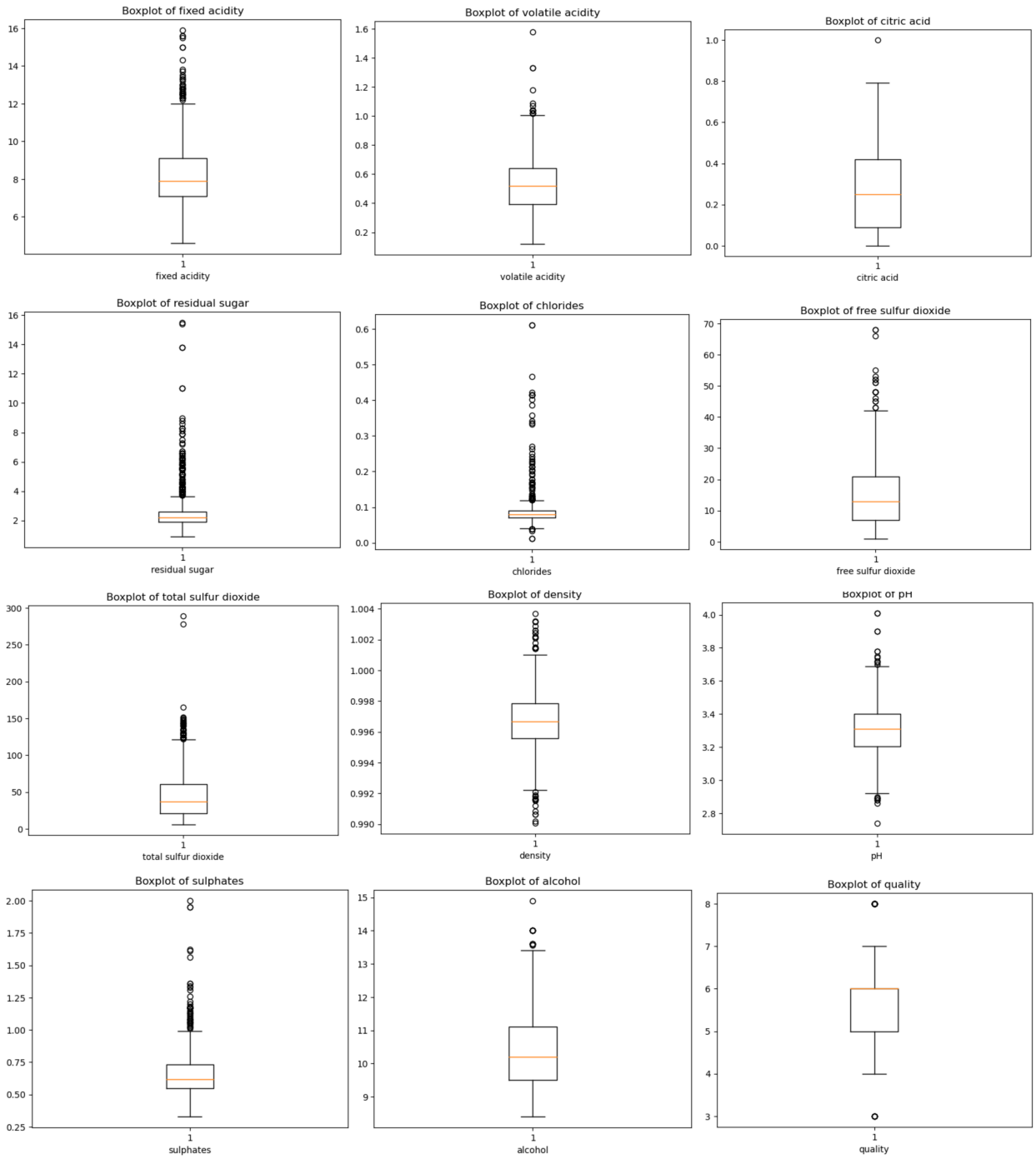


Figure 2A. Box Plot of Each Columns

In Figure 2A, multiple circles representing outliers are noticeable across all columns. To address this issue and enhance the dataset's robustness. I employed an outlier detection method using Interquartile Range (IQR). This method allows us to focus on the middle 50% of the data distribution (between 1st and 3rd quartile). This method is implemented in the `drop_outliers()` function. The function accepts dataframes as an input and will return a new dataframe called `df_clean`.

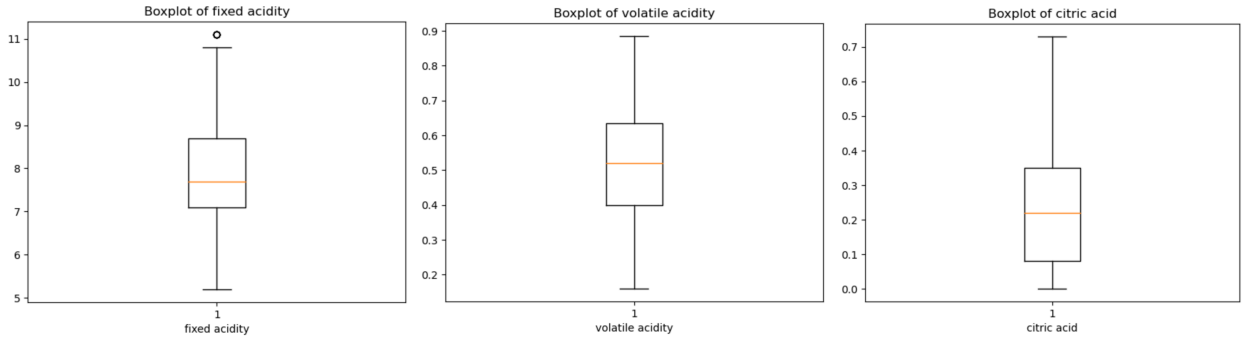
```
In [10]: # DROP Outliers for all columns using IQR
def drop_outliers(df):
    df_new = pd.DataFrame()

    for col in df.columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        IQR = q3 - q1
        lower_bound = q1 - 1 * IQR  # 1.5 and 2 didn't work; 1 worked the best for threshold
        upper_bound = q3 + 1 * IQR  # 1.5 and 2 didn't work; 1 worked the best for threshold

        # drop outliers
        df_new[col] = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)][col]

    return df_new

df_clean = drop_outliers(df)
df_clean = df_clean.dropna()
plt_boxplots(df_clean)
```

Figure 2B. Code Block - Drop Outliers for All Columns Using IQR

This is the general breakdown of the outlier removal process applied to each column:

1. Calculate the 1st and 3rd quartiles
2. Calculate the IQR using the 1st and 3rd quartiles
3. Set the lower and upper bounds for outlier detection using threshold of 1
    a. The outliers are values above the upper bound and below the lower bound
    b. *Note*: through experimentation with different threshold values (1.5 and 2), I found that a threshold value of 1 was most effective in successfully eliminating outliers from the dataset
4. I filtered the dataframe to retain only the values within the upper and lower bounds

After applying the `drop_outliers()` function and filtering the dataframe to remove outliers, the updated boxplot visualization in Figure 2C illustrates the refined distribution and reduced variability of the dataset.



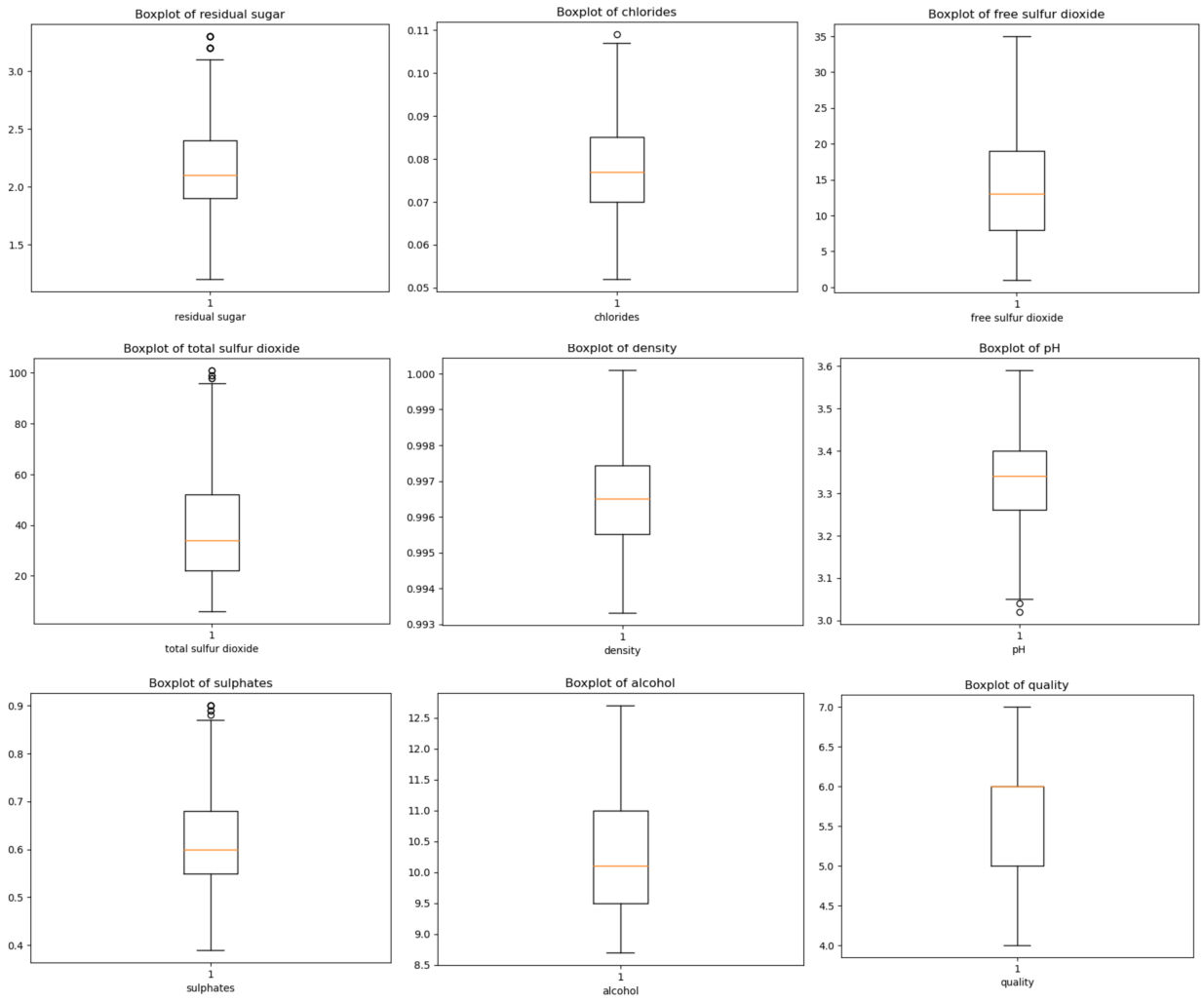


Figure 2C. Box Plot of Each Columns - After drop_outlier() Filter

Below, Figures 2D displays the updated dataframe `df_clean`, showcasing its data structure and dimensions. `df_clean` now consists of 633 columns and 12 rows. We will now be utilizing `df_clean` for analysis and modeling.

```
In [11]: df_clean.head()
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5.0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5.0
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5.0

```
In [12]: df_clean.shape
Out[12]: (633, 12)
```

Figure 2D. `df_clean` Data Structure and Shape

```
# heatmap to see the correlation of each variables
correlation = df_clean.corr()

plt.figure(figsize = (20, 10))
sns.heatmap(correlation, annot=True, linewidths=0.5)
plt.title('Heatmap of Feature Correlation')
```

```
Text(0.5, 1.0, 'Heatmap of Feature Correlation')
```

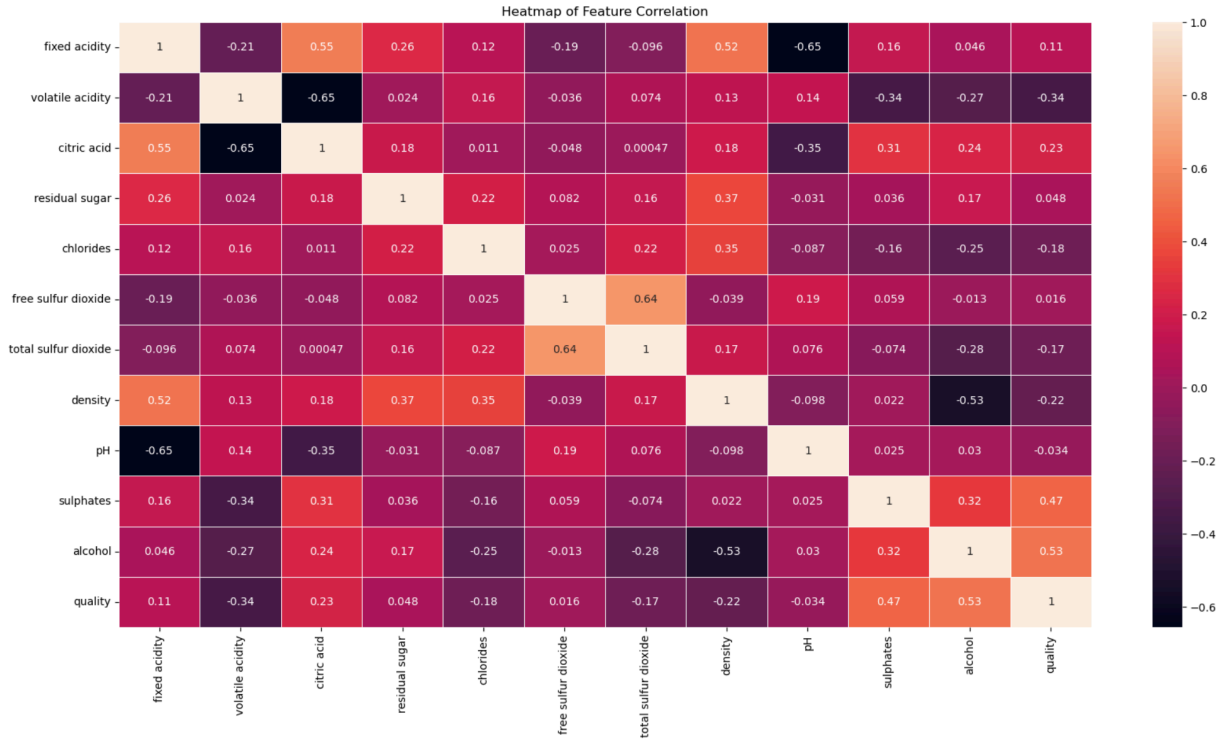


Figure 2E. `df_clean` Heatmap

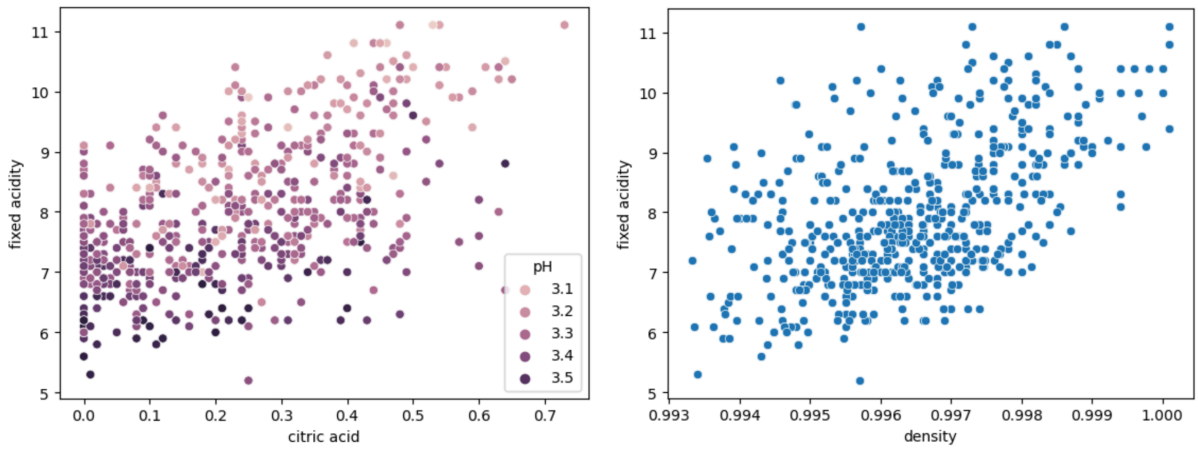


Figure 2F. Strong Negative Positive Heatmaps

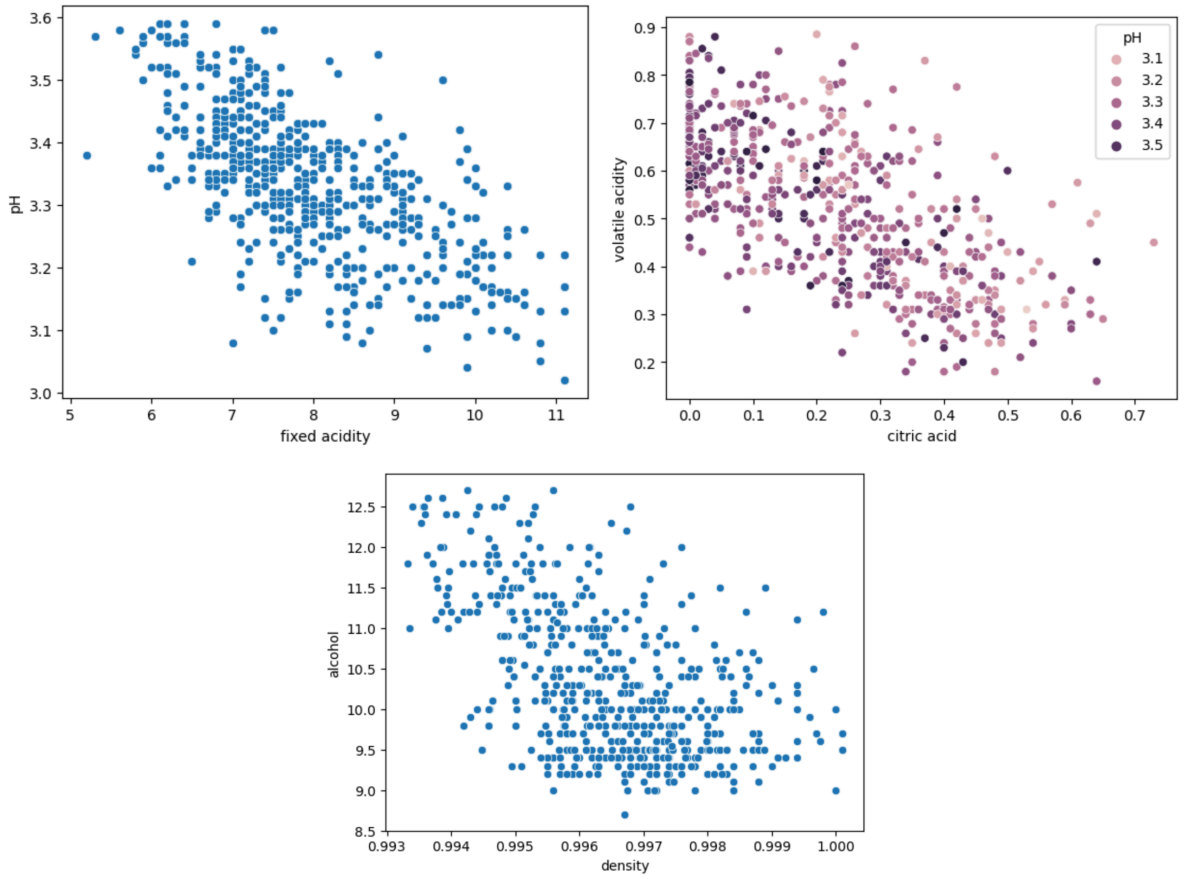


Figure 2G. Strong Negative Correlation Heatmaps

In Figure 2E, a heatmap analysis was conducted to examine the correlation distribution among features in the dataset. Positive correlations highlight relationships where features tend to increase or decrease together, while negative correlations indicate an inverse relationship between features. Here are the key observations and explanations derived from the correlation analysis:

1. **Positive Correlation: Features Strongly Correlated with Target Variable `quality`**
    a. Alcohol vs Quality = 0.53
    b. Sulphates vs Quality = 0.47
    c. Citric Acid vs Quality = 0.23 *(not that strong but better than other values)*

2. **Positive Correlation: Features with Strong Correlation**
    a. Citric Acid vs Fixed Acidity = 0.55
        i. Figure 2F illustrates that if pH levels increases then there's lower citric acid and fixed acidity, and vice versa
        ii. When pH levels increase (lower acidity) then there would be a decrease of citric acid and fixed acidity, and vice versa

    b. Density vs Fixed Acidity = 0.52

i. The positive correlation reflects the influence of dissolved acids on the mass of the wine. Higher concentrations of fixed acids lead to increased density due to the greater presence of dissolved solids in the liquid.

c. Free Sulfur Dioxide vs Total Sulfur Dioxide = 0.64
   i. Chemical equilibrium: SO2 (total) = SO2 (free) + SO2 (bound)
   ii. If SO2 (free) increases, then SO2 (total) would increase as well.

3. Negative Correlation: Features with Negative Correlation

a. Fixed Acidity vs pH = -0.65
   i. Figure 2G demonstrates that as fixed acidity increases, pH decreases; vice versa
   ii. This relationship is a reflection of the acid-base chemistry
   iii. Higher concentrations of acid (fixed acidity) leads to more Hydrogen ions and lower pH (higher acidity), while lower concentrations of acids result in fewer Hydrogen ions and higher pH (lower acidity).

b. Citric Acid vs Volatile Acidity = -0.65
   i. In Figure 2G: citric acid increases, volatile acidity decreases; vice versa

c. Alcohol vs Density = -0.53
   i. Figure 2G demonstrates that as density increases, alcohol decreases; vice versa
   ii. Alcohol is less dense than water and sugar, when yeast converts sugar into alcohol during fermentation, alcohol content increases. Therefore, density decreases.

These correlation insights provide valuable context for understanding the interrelationships among different wine attributes, which can aid in feature selection, model development, and interpretation of results.

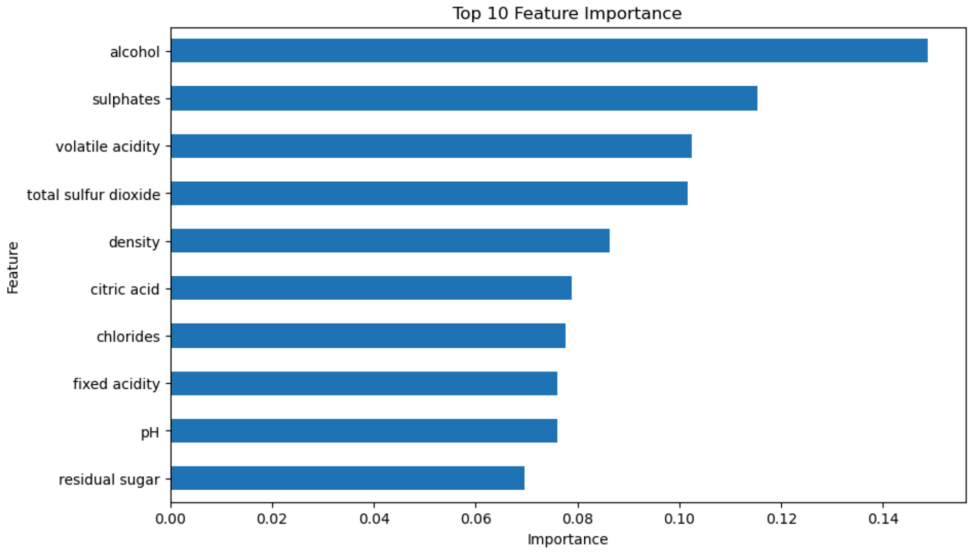


Figure 2H. Feature Importance using Random Forest - Top 10 Features of Importance

I also performed feature importance analysis using Random Forest to identify the top 10 features that significantly contribute to the target column. The outcomes of this analysis are showcased in Figure 2H, providing valuable insights into key attributes that influence the target variable. This aided in shedding insights for feature selection in my modeling process.

Modeling - Feature Selection/Engineering

In the process of building my predictive model for wine quality, I explored various combinations of features to identify the most effective predictors. Leveraging insights from the heatmap analysis, scatterplots, and feature importance derived from Random Forest, I conducted extensive experimentation to optimize my model's predictive performance.

I experimented with different feature combinations, including:

- Features exhibiting strong positive correlation with the target variable
- Features showing strong negative correlation
- Features demonstrating strong positive correlation with other features strongly correlated with the target variable
- Top 10, Top 9, …, Top 1 Features of Importance
- Mix-match

By testing these combinations, I aimed to uncover the optimal set of predictors for my model. Ultimately, the top-performing combination comprised the top 3 features selected through feature selection, as illustrated in Figure 3, representing the most impactful attributes for predicting wine quality with optimal accuracy and reliability.

```
# using TOP 3 FEATURES FROM FEATURE IMPORTANCE <- 72% in RF, 74% in ExtraTrees
X_clean = df_clean[['alcohol', 'sulphates', 'volatile acidity']]
y_clean = df_clean['quality']

X_train_clean, X_test_clean, y_train_clean, y_test_clean = train_test_split(X_clean, y_clean,
                                                                            test_size = 0.2, random_state = 42)
```

Figure 3. Modeling - Feature Engineering

Modeling

Being that this was a classification problem, I've decided to create different models. Some were the same models but with different parameters. For instance, these were the models I've used:

- Logistic Regression (max_iter = 1000, random_state = 42)
- Logistic Regression (random_state = 42)
- Decision Trees (random_state = 42, criterion = gini)
- Decision Trees (random_state = 42)
- Random Forest (n_estimators = 200, random_state = 42)
- Random Forest (random_state = 42)

- Gradient Boosting (n_estimators = 200, learning_rate = 0.1, random_state = 42)
- KNN (n_neighbors = 5)
- KNN (n_neighbors = 10)
- Extra Trees (n_estimators = 200)
- SGD Classifier (loss = 'hinge', random_state = 42)
- SGD Classifier (loss = 'modified_huber', random_state = 42)

Extra Trees (aka. Extremely Randomized Tree) is a type of ensemble learning method based on decision trees. Similar to Random Forest, Extra Trees introduces additional randomness during the tree-building process. Unlike Random Forest, which selects the best feature for splitting at each node, Extra Trees randomly selects feature subsets for splitting nodes. This introduces additional randomness, making the trees more diverse. Additionally, Extra Trees randomly chooses split points for each feature, regardless of the actual optimal split points. This further increases diversity in the trees.

Model Evaluation

Figure 4 presents the outcomes of my modeling efforts. Specifically it displays the accuracy scores achieved by each model listed in the *Modeling* section.

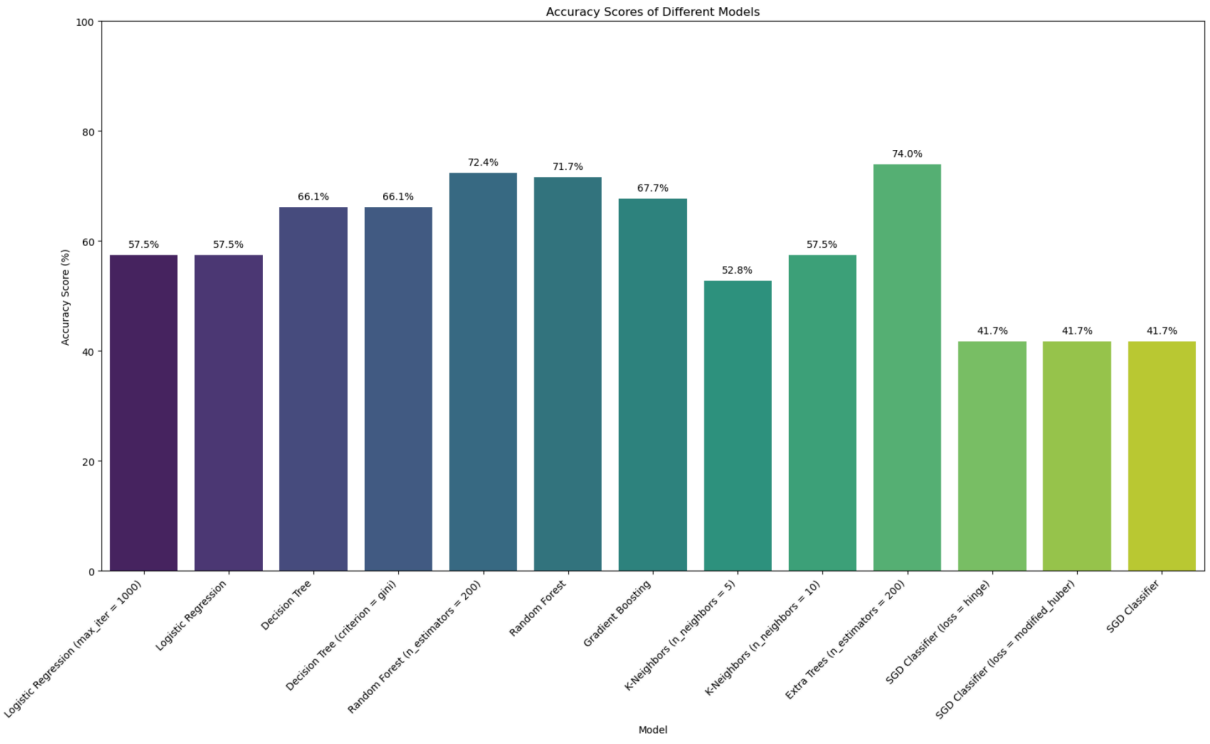


Figure 4. Modeling - Accuracy Scores for each Model

As shown in Figure 4, the top 3 performers were:

1. Extra Trees (n_estimators = 200) - 74%
2. Random Forest (n_estimators = 200, random_state = 42) - 72.4%

3. Random Forest (random_state = 42) - 71.7%

I believe the performance of Extra Trees was the best due to its unique characteristics as an ensemble learning method. Similar to Random Forest, but with distinct differences in how it constructs decision trees. Extra Trees introduce additional randomness during tree construction, which helps reduce variance and improve generalization. The utilization of a higher number of estimators likely enabled the model to capture more nuanced patterns and mitigate overfitting, resulting in a higher accuracy score. Being that Extra Trees is similar to Random Forest, it contributes to why Random Forest performed 2nd best.

Conversely, the SGDClassifier performed poorly, likely due to the dataset's characteristics or the complexity of the wine quality prediction taste. SGD is designed to optimize linear classifiers, which might be less suitable for this dataset if the relationships between features and target are non-linear or require more complex decision boundaries.

Overall, the superior performance of Extra Trees and Random Forest models can be attributed to their ability to handle complex data relationships, reduce overfitting through ensemble learning, and effectively capture relevant patterns in the dataset. On the other hand, the poorer performance of SGD Classifier may stem from its inherent limitations or suboptimal parameter settings for the specific task.

Next Steps

The achieved accuracy of 74% can certainly be improved further with additional refinement and exploration. Given more time, I would pursue several enhancements to enhance model performance:

1. **Feature Engineering Techniques**: implementing advanced feature engineering methods such as scaling, more effective handling of outliers, and exploring new feature creation could lead to improved model accuracy and robustness

2. **Hyperparameter tuning with GridSearchCV**: conducting hyperparameter tuning using techniques like GridSearchCV would allow systematic exploration of optimal model configurations, potentially enhancing predictive performance and generalization.

3. **Experimentation with Different Models**: exploring a wider range of model types beyond those tested could uncover alternative algorithms that offer superior performance for the dataset. Considering more advanced ensemble techniques such as stacking, would be particularly promising given the success of ExtraTrees--an ensemble method--in achieving the highest accuracy

    a. **Stacking**: an advanced modeling technique that involves combining predictions from multiple base models (learners) with different algorithms. A meta-model is then trained to learn how to best combine these predictions, potentially improving overall model performance and robustness.

Conclusion

This project can be accessed through my GitHub Repository: Wine Classification Project